Distributted SPMV algorithm design and performance analysis

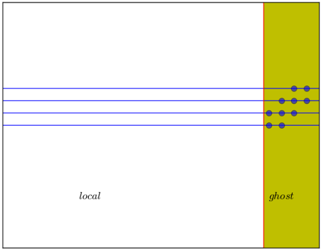
Zhengjiang Li

Department of Mechanical and Aerospace Engineering
University at Buffalo, State University of New York

June 15, 2015

Data Structure

- Distributed Vector
- Distributed Sparse Matrix

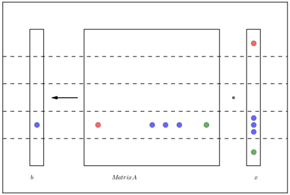


Distributed SPMV

Algorithm 1 SPMV

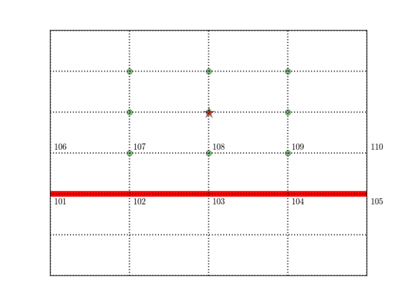
for $i = 0; i < LocalNumberOfRows; i++$ **do**
    $cur\_mat\_val = matrixValues[i]$
    $ur\_local\_ind = matrixInd[i]$
    $cur\_nnz = nnzInRow[i]$
    for $j = 0; j < cur_nnz; j++$ **do**
        $sum+ = cur\_mat\_val[j] * vec\_val[cur\_nnz[j]]$
    end for
end for

spmv



Communication Buffer

GlobalRow	nnzColInCurRow	Interface Communicate
101	(96, 97, 101, 102, 106, 107)	(j, 106, 107)
102	(96, 97, 98, 101, 102, 103, 106, 107, 108)	(j, 106, 107, 108)
103	(97, 98, 99, 102, 103, 104, 107, 108 109	(j, 107, 108, 109)
...	... ...	...



CG algorithm

Algorithm 2 CG

$SparseMatrix A$
$Vector x0, Ax0, b, r, d$
$spmv(A, x0)$
$ScalarProduct(b, Ax0, r)$ % initial residual
$DotProduct(r, r, normr)$; % residual norm
$norm0 = norm$
for $i = 1; i < max\_iter \&\& normr/norm0 > eps; i++$ **do**
  $beta = norm/norm0;$
  $ScalarProduct(z, beta, d, d)$ %update direction vector
  $spmv(A, d, Ad)$
  $DotProduct(d, Ad, dAd);$
  $alpha = norm/dAd;$
  $ScalarProduct(x, alpha, d, x)$; %update solution vector
  $ScalarProduct(r, -alpha, Ad, r)$; %update residual vector
  $DotProduct(r, r, normr)$
end for

Time Complexity

topology	localX	localY	buffer
1D	NX	NY/p	2NX
2D	NX/px	NY/py	2(NX/px + NY/py)
...	... ...	...	

CPU calcuation time :

$$t_1 = (2m + 2N)/c_1 = 2(nnzInRow + 1)N/c_1 \tag{1}$$

Communicate time:

$$t_2 = 2NX/c_2 \% 1D\ cpu \tag{2}$$

$$t_2 = 2(NX/px + NY/py)/c_2 \% 2D\ cpu \tag{3}$$

Time Complexity II

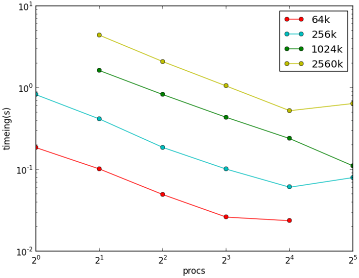
$$t = t_1 + t_2 = \begin{cases} 2\delta \frac{NX \cdot NY}{p \cdot c_1} + 2\frac{NX}{c_2} \ \%1D\ cpu \\ 2\delta \frac{NX \cdot NY}{p \cdot c_1} + 2\frac{NX \cdot py + NY \cdot px}{px \cdot py \cdot c_2} \ \%2D\ cpu \end{cases}$$

The time complexity is proportional to the size of problem, nonzeros In rows, as well as hardward performance. The speed-up will reach an upper limit when:

1. For 1D CPU topology, $p \geq \gamma NX$
2. For 2D CPU topology, $2\sqrt{p} \geq \gamma NX$

Also consider accuracy(iteration numbers)

Results



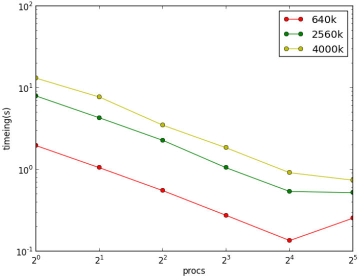


Figure : CPU performance

sparse matrix src

```
struct SparseMatrix
{
  Geometry*  geom;

  /*local variable */
  local_int_t localNumberOfRows;
  local_int_t localNumberOfColumns;
  local_int_t localNumberOfNonzeros;
  global_int_t** mtxIndG;
  global_int_t** mtxIndL;
  double** matrixValues;

  global_int_t* nonzerosInRow;

  std::map<global_int_t, local_int_t> globalToLocalMap;
  std::vector<global_int_t> localToGlobalMap;

```

Thank you for your attention [1]

[1] **Jonathan Richard Shewchuk "An introduction to the Conjugate Gradietn Metho**